

Java CM4

Olivier Marchetti

Laboratoire d'informatique de Paris 6 – Pôle SoC – Sorbonne Université

22 novembre 2021



Plan

- ❶ Les Interfaces Utilisateurs en JAVA
 - Introduction
 - Composants SWING
 - Composants internes
 - Dessiner
- ❷ Les gestionnaires de répartition
 - FlowLayout
 - BorderLayout
 - GridLayout
 - GridBagLayout
 - CardLayout
- ❸ Introduction à la programmation événementielle
- ❹ Interfaces pour la programmation événementielle
 - *ActionListener*
 - *Window[Focus]Listener*
 - *KeyListener*
 - *Mouse[Motion/Wheel]Listener*
 - *ItemListener*



1 Les Interfaces Utilisateurs en JAVA

- Introduction
- Composants SWING
- Composants internes
- Dessiner

2 Les gestionnaires de répartition

- FlowLayout
- BorderLayout
- GridLayout
- GridBagLayout
- CardLayout

3 Introduction à la programmation événementielle

4 Interfaces pour la programmation événementielle

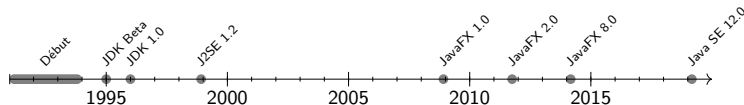
- *ActionListener*
- *Window[Focus]Listener*
- *KeyListener*
- *Mouse[Motion/Wheel]Listener*
- *ItemListener*

Un aperçu

JAVA propose une large gamme de classes permettant de mettre au point des interfaces conviviales :

- ▶ applications conformes à la philosophie *"Write Once, Run Anywhere"*.
- ▶ utilisant :
 - ① les concepts de la POO,
 - ② la programmation dite « événementielle » ,
 - ③ les techniques de génie logiciel (patrons de conception/*design patterns*).

Depuis sa création, JAVA a subi quelques évolutions pour ses interfaces utilisateurs :



Quelques jalons :

- ▶ 1996 – JDK1 : composant graphiques AWT.
- ▶ 1998 – J2SE 1.2 : composants graphiques SWING.
- ▶ 2008 – JavaFX 1.0 : nouvelles bibliothèques... propriétaires et fermées.
- ▶ 2011 – JavaFX 2.0 : ORACLE ouvre le code de JAVA FX.
- ▶ 2014 – JavaFX 8.0 : intégration pleine dans JAVA 8.

AWT, SWING et JAVAFX – 1/3

- ▶ Le paquetage AWT, *Abstract Window Toolkit*, fut le premier système d'interfaces utilisateurs de JAVA :
 - il repose sur le système de fenêtrage de la machine hôte (utilise des composants natifs, dits lourds) ;
 - manque d'uniformité graphique ;
 - possibilité restreinte par les systèmes de fenêtrage des OS.
- ▶ Le paquetage SWING repose en partie sur AWT mais :
 - dispose de ses propres composants (dits légers),
 - enrichit considérablement les possibilités offertes,
 - possibilité de spécifier un « *look and feel* » (rendu graphique),
 - uniformise le rendu graphique quelque soit l'OS.
- ▶ Le paquetage JAVAFX repense la conception des interfaces utilisateurs et la fait évoluer pour tenir compte des terminaux mobiles.

✗ Hélas, ANDROID est passé par là...

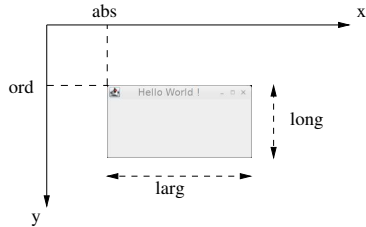
✓ SWING n'est plus développé mais est toujours maintenu !

Un premier exemple avec SWING – 1/2

Ce programme ouvre une fenêtre munie des trois boutons standards :

```
import javax.swing.*;

class MaPremiereFenetre {
    public static void main(String args[]) {
        JFrame maFenetre = new JFrame("Hello World !");
        maFenetre.setBounds(100, // abs
                           400, // ord
                           300, // larg
                           150); // long
        maFenetre.setVisible(true);
    }
}
```



Remarques :

- ▶ Instanciation avec l'instruction `new JFrame("Titre")`.
- ▶ La méthode `setBounds()` permet :
 - 1 de disposer précisément la fenêtre sur l'écran ;
 - 2 de définir les dimensions de cette fenêtre.
- ▶ La fenêtre est rendue visible avec la méthode `setVisible()`.

Un premier exemple avec SWING – 2/2

Il faut distinguer :

- ▶ le fonctionnement du programme ;
- ▶ de son affichage graphique.

Notamment, le bouton de fermeture de la fenêtre fermera la fenêtre sans mettre fin au programme.

✓ Sous GNU-LINUX, taper CTRL-C pour tuer le programme.

On peut améliorer ce programme en ajoutant une action sur ce bouton :

```
import javax.swing.*;

class MaFenetreAvecTerminaison {
    public static void main(String args[]) {
        JFrame maFenetre = new JFrame("Hello World !");
        maFenetre.setBounds(100, 400, 300, 150);
        maFenetre.setVisible(true);
        maFenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



La classe JFrame

Un objet de type JFrame¹ est un objet qui hérite de classes issues de AWT :

```
[15:36][Prog pc666 :]$ java Ascendance javax.swing.JFrame
java.lang.Object
  \_java.awt.Component
    \_java.awt.Container
      \_java.awt.Window
        \_java.awt.Frame
          \_javax.swing.JFrame
```

Ce composant « lourd » dispose d'un conteneur et hérite de méthodes utiles :

► Jouer sur le titre :

```
void setTitle("Titre");
String getTitle();
```

► Jouer sur la taille :

```
void setSize(longueur, largeur);
Dimension getSize();
void setResizable(true/false);
```

► Jouer sur son conteneur :

```
Container getContentPane();
void setContentPane(Container c);
```

► Positionner :

```
Point getLocation();
Point getLocationOnScreen();
void setLocation(x, y);
Toolkit getToolkit(); // Infos écran.
```

► Jouer sur le *focus* :

```
boolean isFocusable();
void setFocusable(true/false);
boolean isFocused();
void requestFocus();
```

¹<https://docs.oracle.com/javase/8/docs/api/javax/swing/JFrame.html>

La classe JFrame – exemple

Réalisation avec héritage d'une fenêtre centrée sur l'écran avec un dimensionnement précis et une couleur de fond :

```
import javax.swing.*;
import java.awt.*;

class MaFenetreCentree extends JFrame {
    public MaFenetreCentree(String titre) {
        setTitle(titre);
        getContentPane().setBackground(Color.RED);
        int taille = 500;
        setPreferredSize(new Dimension(taille, taille));
        Dimension dimEcran = Toolkit.getDefaultToolkit().getScreenSize();
        setLocation(new Point( (dimEcran.width - taille) / 2,
                               (dimEcran.height - taille) / 2 ));
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pack();
        setVisible(true);
    }
}

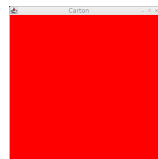
class JFrameHeritage {
    public static void main(String args[]) {
        MaFenetreCentree fc = new MaFenetreCentree(args[0]);
    }
}
```

Couleur de fond.

Informations sur l'écran.

Positionnement final.

Calcul des dimensions.



Apparence d'une interface utilisateur – composants lourds vs. légers (1/3)

Tout objet graphique hérite de la classe abstraite `java.awt.Component`¹.

► Distinction entre composants :

- `awt` dit lourds/natifs \approx fenêtres;
- `swing` dit légers \approx internes.

► Méthodes pour :

- modifier l'apparence générale

```
// En plus du dimensionnement :  
Color getForeground();  
void setForeground(Color c);  
Color getBackground();  
void setBackground(Color c);  
void repaint();
```

► Attributs statiques de disposition
`XXX_ALIGNMENT` où

$XXX \in \{\text{BOTTOM/TOP, CENTER, LEFT/RIGHT}\}$

- initier un tracé/dessin

```
// Tracé quelconque :  
Graphics getGraphics();  
void dispose();  
void setColor();  
void getColor();  
// Tracé écrit :  
void getFont(Color c);  
void setFont(Font f);
```

L'objet `Graphics`² : contexte graphique d'un composant

Il désigne les ressources mobilisables pour effectuer des tracés :

- | | |
|--------------------|---|
| ① géométriques : | \implies <code>drawLine()</code> , <code>drawOval()</code> , <code>drawRect()</code> ... |
| ② écrits | \implies <code>getFont()</code> , <code>getFontMetrics()</code> , <code>drawString()</code> ... |
| ③ gestion d'images | \implies <code>drawImage()</code> ... |

¹<https://docs.oracle.com/javase/8/docs/api/java/awt/Component.html>

²<https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics.html>

Apparence d'une interface utilisateur - les couleurs en JAVA (3/3)

La classe `java.awt.Color`¹ permet de régler et définir les couleurs.

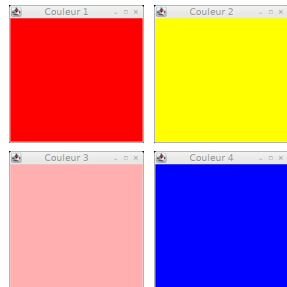
```
import javax.swing.*;
import java.awt.*;
import static java.awt.Color.*;

class MonNuancier {
    public static void main(String args[]) {
        int incrementX = 310;
        int incrementY = 355;
        Color tabCouleurs[][] = {
            {RED, YELLOW},
            {PINK, BLUE}
        };
    };
    int cptCouleur = 0;
    for (int i = 0; i < tabCouleurs.length; i++) {
        for (int j = 0; j < tabCouleurs[0].length; j++) {
            cptCouleur++;
            JFrame f = new JFrame("Couleur " + cptCouleur);
            f.getContentPane().setBackground(tabCouleurs[i][j]);
            f.setPreferredSize(new Dimension(300, 300));
            f.setLocation(j * incrementX,
                i * incrementY);

            f.pack();
            f.setVisible(true);
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
    }
}
```

- ▶ Nombreuses couleurs prédéfinies (en `static`);
- ▶ Nombreux constructeurs (format RGB...);
- ▶ Nombreuses méthodes pour agir dessus.

À l'exécution,



¹<https://docs.oracle.com/javase/8/docs/api/java/awt/Color.html>

Apparence d'une interface utilisateur – le « *look and feel* » 3/3

Selon votre JVM, il est possible de modifier l'apparence des composants :

```
import javax.swing.*;
import java.awt.*;

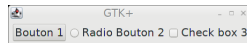
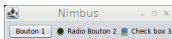
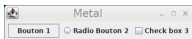
class ListerLookAndFeel {
    public static void main(String args[]) throws ClassNotFoundException,
        InstantiationException,
        IllegalAccessException,
        UnsupportedLookAndFeelException {

        int i = 0;
        UIManager.LookAndFeelInfo lookAndFeelDispo[] = UIManager.getInstalledLookAndFeels();
        for (UIManager.LookAndFeelInfo lookAndFeel : lookAndFeelDispo) {
            UIManager.setLookAndFeel(lookAndFeel.getClassName());
            JFrame maFenetre = new JFrame(lookAndFeel.getName());
            maFenetre.setLocation(100 + i * 380, 100);
            i++;
            maFenetre.getContentPane().setLayout(new FlowLayout());
            maFenetre.add(new JButton("Bouton " + 1));
            maFenetre.add(new JRadioButton("Radio Bouton " + 2));
            maFenetre.add(new JCheckBox("Check box " + 3));
            maFenetre.pack();
            maFenetre.setVisible(true);
            maFenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
    }
}
```

Ajuster l'apparence.

Ajouter les composants en ligne.

À l'exécution, sur ma machine :



Le packaging SWING – composants lourds vs. légers

Il¹ contient l'ensemble des composants pour réaliser des interfaces utilisateurs (en abrégé IU).

⇒ tous les noms dans l'API commence par un J.

On distinguera deux types de composants :

► Composants lourds/natifs :

- JDialog,
- JFileChooser,
- JFrame,
- JOptionPane,
- ...

Ce sont des composants de « premier plan », *i.e.* des fenêtres dans lesquelles nous construirons nos IU.

✗ Éviter de faire du dessin.

► Composants légers :

- JPanel
- JScrollPane
- JMenuBar
- JProgressBar
- ...

Ce sont des composants internes, propre à JAVA, avec lesquels les fenêtres seront composées.

✓ Ok pour le dessin.

¹<https://docs.oracle.com/javase/8/docs/api/javax/swing/package-tree.html>

Composants lourds – JOptionPane

Nombreuses méthodes statiques faciles d'emploi et bien documentées dans l'API¹.

```
import javax.swing.*;

class MesJOptionPane {
    public static void main(String args[]) {
        JOptionPane.showMessageDialog(null,
            "Êtes-vous attentif ?",
            "Attention",
            JOptionPane.WARNING_MESSAGE);

        JOptionPane.showConfirmDialog(null,
            "Êtes-vous vraiment certains ?",
            "Attention",
            JOptionPane.YES_NO_CANCEL_OPTION);

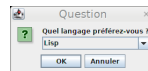
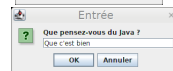
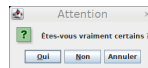
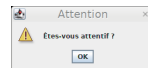
        String reponse = JOptionPane.showInputDialog(null,
            "Que pensez-vous du Java ?");
        System.out.println(reponse);

        String langages[] = {"Ada", "C++", "C", "Lisp", "OCaml",
            "Perl", "Python", "Rust"};

        reponse = (String) JOptionPane.showInputDialog(null,
            "Quel langage préférez-vous ?",
            "Question",
            JOptionPane.QUESTION_MESSAGE,
            null,
            langages,
            langages[3]);

        System.out.println(reponse + "...beurk !");
    }
}
```

Dans l'ordre :



¹<https://docs.oracle.com/javase/8/docs/api/index.html?javax/swing/JOptionPane.html>

Composants lourds – JDialog

Le composant JDialog¹ est une fenêtre de dialogue personnalisable.

```
import javax.swing.*;

class MonJDialog {
    public static void main(String args[]) {
        int taille = 500;
        JFrame fRacine = new JFrame("Monologue");
        fRacine.setSize(taille, taille / 2);
        fRacine.setLocation(taille, taille);
        fRacine.setVisible(true);
        fRacine.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JDialog fDialogue = new JDialog(fRacine,
                                     "ADP : non ou NON ?",
                                     true);

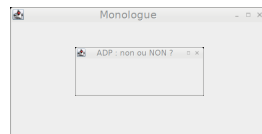
        fDialogue.setSize( ((int) (taille / 1.5f)),
                           taille / 4);
        fDialogue.setLocationRelativeTo(fRacine);
        fDialogue.setVisible(true);
        fDialogue.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    }
}
```

Création d'une fenêtre de dialogue :

- ① associée à fRacine ;
- ② avec un titre ;
- ③ et modale (i.e. capturant toute l'interaction)

Placement relatif et centré.

À l'exécution, nous obtenons :



¹ <https://docs.oracle.com/javase/8/docs/api/javax/swing/JDialog.html>

Composant lourd – JFileChooser

Ce composant¹ permet d'afficher un sélecteur de fichier.

```
import javax.swing.*;
import javax.swing.filechooser.*;

class MonJFileChooser {
    public static void main(String args[]) {
        JFileChooser jfc = new JFileChooser("/home/om/Bureau/CM4_java_Beamer/Prog");
        FileNameExtensionFilter filtre = new FileNameExtensionFilter("Codes sources java",
                                                                    "java");

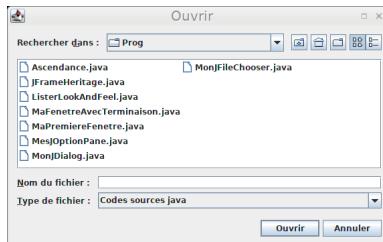
        jfc.setFileFilter(filtre);
        jfc.showOpenDialog(null);
    }
}
```

Remarque :

Création
d'un filtre.



À l'exécution, nous aurons alors :



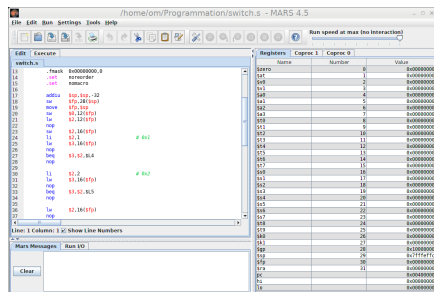
¹<https://docs.oracle.com/javase/8/docs/api/javax/swing/JFileChooser.html>

Les composants légers - zoologie

Les programmes avec IU utilisent souvent des bibliothèques spécialisées telles que QT, GTK+ et même voire SWING¹.

Le simulateur MARS² pour le processeur MIPS :

- ▶ écrit en JAVA avec une IU en SWING ;
- ▶ permet d'éditer du code assembleur ;
- ▶ de l'assembler ;
- ▶ d'en faire le suivi à la trace ;
- ▶ d'examiner les registres, le cache et la mémoire ;
- ▶ de simuler le chemin des données.



¹Écrit en grande partie en JAVA, l'IDE ECLIPSE repose sur la bibliothèque graphique (SWT).

²Disponible gratuitement en ligne.

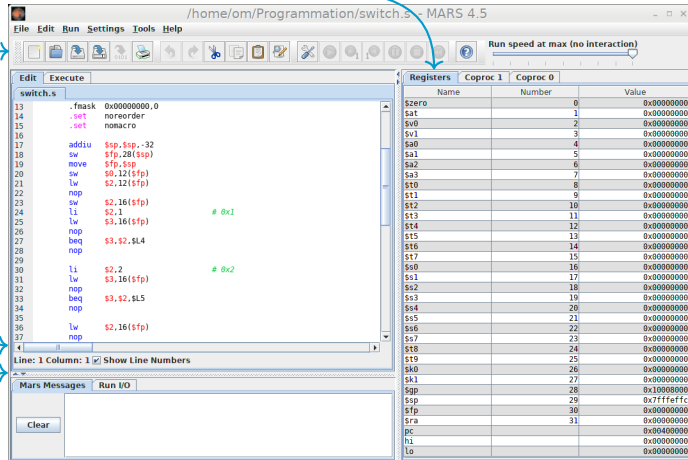
Les composants légers : les composants internes généraux

Onglet avec JTabbedPane

Barre d'outils
JToolBar

Barres ascen-
seurs de fenêtre
JScrollPane

Séparateur avec
JSplitPane



Les composants légers : composants de contrôle – 1/2

Boutons JButton

Barre de menus
JMenuBar

Entrée avec
JMenuItem

Menu avec
JMenu

Boutons
JCheckBox

Réglage avec JSlider

The screenshot shows the MARS 4.5 MIPS simulator window titled "/home/om/Programmation/switch.s - MARS 4.5". The interface includes a menu bar (File, Edit, Run, Settings, Tools, Help), a toolbar, a main assembly code editor, a register window, and a console area. Blue arrows point from text labels to specific UI elements: "Boutons JButton" points to the toolbar; "Barre de menus JMenuBar" points to the menu bar; "Entrée avec JMenuItem" points to the "Exit" menu item; "Menu avec JMenu" points to the "File" menu; "Boutons JCheckBox" points to the "Show Line Numbers" checkbox; and "Réglage avec JSlider" points to the "Run speed at max (no interaction)" slider.

Registers		Coproc 1	Coproc 0
Name	Number		Value
\$zero	0		0x00000000
\$at	1		0x00000000
\$v0	2		0x00000000
\$v1	3		0x00000000
\$a0	4		0x00000000
\$a1	5		0x00000000
\$a2	6		0x00000000
\$a3	7		0x00000000
\$t0	8		0x00000000
\$t1	9		0x00000000
\$t2	10		0x00000000
\$t3	11		0x00000000
\$t4	12		0x00000000
\$t5	13		0x00000000
\$t6	14		0x00000000
\$t7	15		0x00000000
\$s0	16		0x00000000
\$s1	17		0x00000000
\$s2	18		0x00000000
\$s3	19		0x00000000
\$s4	20		0x00000000
\$s5	21		0x00000000
\$s6	22		0x00000000
\$s7	23		0x00000000
\$t8	24		0x00000000
\$t9	25		0x00000000
\$k0	26		0x00000000
\$k1	27		0x00000000
\$gp	28		0x10000000
\$sp	29		0x7fffffc0
\$fp	30		0x00000000
\$ra	31		0x00000000
pc			0x00400000
hi			0x00000000
lo			0x00000000

Les composants légers : composants de contrôle – 2/2

Liste déroulante
JComboBox

Label
JSpinner

Bouton
radio
JRadioButton

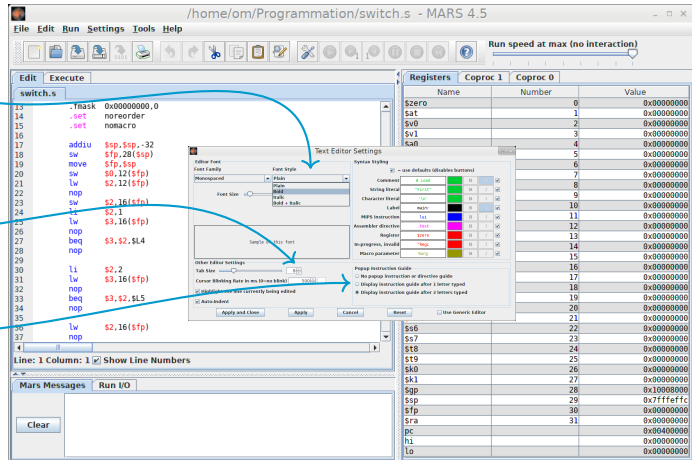
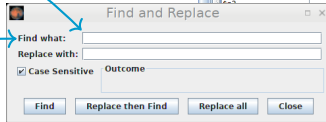


Table avec JTable

Label avec
JLabel

Bulle d'aide avec
JToolTip (survol pointeur
souris)



Composants : synthèse

► Composants de premier niveau :

- JFrame
- JDialog
- JOptionPane
- JFileChooser
- JColorChooser : palette de couleurs.

► Composants de contrôle :

- JButton
- JRadioButton
- JSpinner
- JCheckBox
- JComboBox
- JSlider
- JMenu (avec JMenuBar, JMenuItem)
- JToggleButton : bouton à deux états.
- JPopupMenu : menu-contextuel.
- JList : liste d'éléments avec sélection simple ou multiple.
- JPasswordField : champ pour mot de passe.

► Composants d'organisation :

- JToolBar
- JTabbedPane
- JSplitPane
- JScrollPane : attention, ce composant n'apparaît que si son contenu est très grand.
- JSeparator marqueur de séparation (cf. menu *File* de MARS).
- JPanel : simple panneau, utile pour dessiner.

► Composants d'information :

- JLabel (non-interactif)
- JTextField (interactif)
- JTextArea
- JTable
- JToolTip
- JProgressBar : barre de progression.
- JTree : présentation arborescente.

Un premier dessin

Ici, on dessine directement sur la `JFrame` en redéfinissant la méthode `paint()` :

```
import java.awt.*;
import javax.swing.*;

class DessinJFrame extends JFrame {

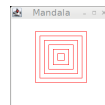
    public void paint(Graphics g) {
        super.paint(g); // On repeint le composant tel que prédéfini.
        float tailleCoteFenetre = (float) (Math.min(getWidth(),
                                                    getHeight()));

        float tailleCote = tailleCoteFenetre / 2;
        float xCoinGauche = tailleCote / 2;
        float yCoinGauche = tailleCote / 2;
        g.setColor(Color.RED);
        for (float i = 0.0f; i < tailleCote / 2 ; i += 10) {
            g.drawRect((int) (xCoinGauche + i),
                      (int) (yCoinGauche + i),
                      (int) (tailleCote - 2 * i),
                      (int) (tailleCote - 2 * i));
        }
    }

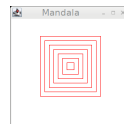
    public static void main(String args[]) {
        DessinJFrame fDessin = new DessinJFrame();
        int tailleCote = 230;
        fDessin.setSize(tailleCote, tailleCote);
        fDessin.setTitle("Mandala");
        fDessin.getContentPane().setBackground(Color.WHITE);
        fDessin.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fDessin.setVisible(true);
    }
}
```

À l'exécution :

► au début,



► après étirement,



La JVM appellera automatiquement la méthode `paint()`.

Un premier dessin... éphémère

Ici, le dessin est défini dans une méthode de notre cru :

```
import java.awt.*;
import javax.swing.*;

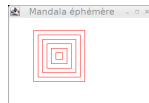
class DessinJFrame extends JFrame {

    void peindre() {
        Graphics g = getGraphics(); // Obtention du contexte graphique.
        float tailleCoteFenetre = (float) (Math.min(getWidth(),
                                                    getHeight()));
        ...
        g.dispose(); // Restitution conseillée de ce contexte graphique.
    }

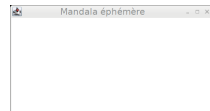
    public static void main(String args[]) throws InterruptedException {
        DessinJFrame fDessin = new DessinJFrame();
        fDessin.setSize(330, 230);
        fDessin.setTitle("Mandala éphémère");
        ...
        fDessin.setVisible(true);
        Thread.sleep(100); // Autrement, il n'y aurait aucun dessin.
        fDessin.peindre();
    }
}
```

À l'exécution :

► au début,



► après étirement,



Conclusion

À l'exécution, la JVM invoquera dès que nécessaire la méthode `paint()` :

- ✓ si elle est redéfinie alors le dessin sera permanent ;
- ✗ sinon tout dessin finira par être effacé ou dégradé (à éviter/proscrire).

Un deuxième dessin avec le composant JPanel

Le composant `JPanel`¹ est un composant SWING qui hérite de classe `Component` :

```
[12:02][Prog pc666 :]$ Ascendance javax.swing.JPanel
java.lang.Object
  \_java.awt.Component
    \_java.awt.Container
      \_javax.swing.JComponent
        \_javax.swing.JPanel
```

Usage :

- Composition d'IU : ce composant héritant de `Container`, il pourra s'ajouter d'autres composants ;
- Dessin/Tracé.

Exemple de dessin permanent :

```
import java.awt.*;
import static java.awt.Color.*;
import javax.swing.*;

class ZoneDessin extends JPanel {

    ZoneDessin() {
        setPreferredSize(new Dimension(500, 500));
        setBackground(new Color(51, 153, 255));
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g); // Repeint le composant.
        Color tabCouleur[] = {RED, WHITE, BLUE};
        int pas = 100, i = 0, XYorigCadre = pas;
        for (int rayon = 300; rayon >= 100; rayon -= pas) {
            g.setColor(tabCouleur[i]);
            g.fillOval(XYorigCadre, XYorigCadre, rayon, rayon);
            i++;
            XYorigCadre += pas / 2;
        }
    }
}
```

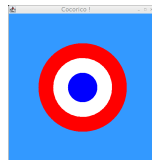
```
public static void main(String args[]) {
    JFrame f = new JFrame("Cocorico !");
    f.getContentPane().add(new ZoneDessin());
    f.pack();
    f.setVisible(true);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```



À l'exécution :

Remarques :

Rôle analogue à `paint()` pour `JComponent`.



¹<https://docs.oracle.com/javase/8/docs/api/javax/swing/JPanel.html>

Un troisième dessin avec une image

```
import javax.swing.*;
import java.awt.*;

class Portrait extends JPanel {
    Image image;

    Portrait() {
        String cheminJPG = "../Figure/AnnickAlexandre.jpg";
        image = getToolkit().getImage(cheminJPG);
        setPreferredSize(new Dimension(500, 500));
        setBackground(Color.BLUE);
        setOpaque(true);
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g); // Commentez cette ligne.
        g.setFont(new Font("SansSerif",
                           Font.BOLD + Font.ITALIC,
                           40));
        ((Graphics2D) g).setStroke(new BasicStroke(10));
        g.setColor(Color.RED);
        g.drawImage(image, 100, 75, 300, 300, this);
        g.drawRect(100, 75, 300, 300);
        g.drawString("Mme. Alexandre", 65, 450);
    }

    public static void main(String args[]) {
        JFrame maFenetre = new JFrame("Qui commande ?");
        maFenetre.setContentPane(new Portrait());
        maFenetre.pack();
        maFenetre.setVisible(true);
        maFenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

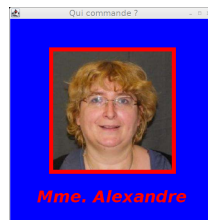
Lorsque l'interface se modifie (suite à un étirement, ouverture d'une liste...), tout ou partie des composants de l'interface doit se redessiner. Appel automatique de :

- ▶ `repaint()` pour les composants lourds (avec appel implicite de `paint()`);
- ▶ `paintComponent()` pour les composants légers.

Pour le dessin, on redéfinira cette dernière méthode et invoquera au début le dessin du composant lui-même.

Remarques :

- Réglages d'une fonte et de l'épaisseur du trait.
- Dessins (image, rectangle, message).



1 Les Interfaces Utilisateurs en JAVA

- Introduction
- Composants SWING
- Composants internes
- Dessiner

2 Les gestionnaires de répartition

- FlowLayout
- BoxLayout
- BorderLayout
- GridLayout
- GridBagLayout
- CardLayout

3 Introduction à la programmation événementielle

4 Interfaces pour la programmation événementielle

- *ActionListener*
- *Window[Focus]Listener*
- *KeyListener*
- *Mouse[Motion/Wheel]Listener*
- *ItemListener*

Gestionnaire de répartition – intérêt

Pour réaliser une interface graphique, il faut assembler et imbriquer des composants entre eux.

Un composant pouvant contenir d'autres composants hérite de la classe `Container` :

```
[15:55][Prog pc666 :]$ Ascendance javax.swing.JPanel
java.lang.Object
  \_java.awt.Component
    \_java.awt.Container
      \_javax.swing.JComponent
        \_javax.swing.JPanel
```

```
[15:55][Prog pc666 :]$ java Ascendance javax.swing.JFrame
java.lang.Object
  \_java.awt.Component
    \_java.awt.Container
      \_java.awt.Window
        \_java.awt.Frame
          \_javax.swing.JFrame
```

Tout conteneur dispose d'un « *layout* » .

Un tel objet permet d'ajouter les composants facilement selon un modèle de disposition prédéfini.

⇒ facilite la conception.

Gestionnaire de répartition – quelques méthodes

- Pour accéder/modifier cet objet :

```
LayoutManager getLayout();  
void setLayout(LayoutManager);
```

- Pour redéfinir le gestionnaire d'un conteneur :

```
setLayout(new XXXLayout);
```

avec XXXLayout pris parmi notamment :

- FlowLayout
- BorderLayout
- GridLayout
- GridBagLayout
- CardLayout

tous implémentant l'interface `LayoutManager`¹.

- Pour ajouter un composant à un objet conteneur :
- Mise-à-jour dynamique de l'interface :

```
add(Composant);
```

```
validate();  
remove([Component] c);  
removeAll();
```

¹<https://docs.oracle.com/javase/8/docs/api/java/awt/LayoutManager.html>

Le gestionnaire FlowLayout

Ce gestionnaire¹ :

- ▶ dispose en ligne les composants avec la méthode `add()` ;
- ▶ est celui par défaut de certains composants (e.g. `JPanel`).

```
import javax.swing.*;
import java.awt.*;

class PanneauInterieur extends JPanel {
    PanneauInterieur() {
        setLayout(new FlowLayout(FlowLayout.CENTER, 20, 20));
        add(new JLabel("Voulez-vous poursuivre ?"));
        add(new JButton("Oui"));
        add(new JButton("Non"));
    }
}

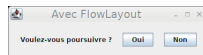
class ExempleFlowLayout {
    public static void main(String args[]) {
        JFrame maFenetre = new JFrame("Avec FlowLayout");
        maFenetre.setContentPane(new PanneauInterieur());
        maFenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        maFenetre.pack();
        maFenetre.setVisible(true);
    }
}
```

Redéfinition du gestionnaire :

- centré,
- bourrage.

Ajouts successifs des composants.

À l'exécution :



¹<https://docs.oracle.com/javase/8/docs/api/java/awt/FlowLayout.html>

Le gestionnaire BorderLayout

Ce gestionnaire¹ dispose les composants en colonne ou en ligne :

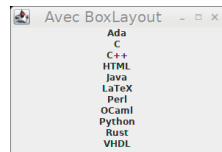
```
import javax.swing.*;
import java.awt.*;

class PanneauInterieur extends JPanel {
    PanneauInterieur() {
        setLayout(new BorderLayout(this, BorderLayout.Y_AXIS));
        String tabLangages[] = {"Ada", "C", "C++", "HTML", "Java",
                                "LaTeX", "Perl", "OCaml", "Python",
                                "Rust", "VHDL"};
        for (String langage : tabLangages) {
            JLabel label = new JLabel(langage);
            label.setAlignmentX(Component.CENTER_ALIGNMENT);
            add(label);
        }
    }
}

class ExempleBoxLayout {
    public static void main(String args[]) {
        JFrame maFenetre = new JFrame("Avec BorderLayout");
        maFenetre.setContentPane(new PanneauInterieur());
        maFenetre.setPreferredSize(new Dimension(300, 200));
        maFenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        maFenetre.pack();
        maFenetre.setVisible(true);
    }
}
```

Redéfinition du gestionnaire.

Positionnement des Label.



¹<https://docs.oracle.com/javase/8/docs/api/javax/swing/BoxLayout.html>

Le gestionnaire BorderLayout

Ce gestionnaire¹ dispose les composants selon les directions cardinales :

```
import javax.swing.*;
import java.awt.*;

class PanneauInterieur extends JPanel {
    PanneauInterieur() {
        setLayout(new BorderLayout(20, 40));
        add(new JLabel("Au nord", JLabel.CENTER),
            BorderLayout.NORTH);
        add(new JLabel("À l'ouest", BorderLayout.WEST);
        add(new JLabel("Au centre", JLabel.CENTER),
            BorderLayout.CENTER);
        add(new JLabel("À l'est", BorderLayout.EAST);
        add(new JLabel("Au sud", JLabel.CENTER),
            BorderLayout.SOUTH);
    }
}

class ExempleBorderLayout {
    public static void main(String args[]) {
        JFrame maFenetre = new JFrame("Avec BorderLayout");
        maFenetre.setContentPane(new PanneauInterieur());
        maFenetre.setPreferredSize(new Dimension(350, 150));
        maFenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        maFenetre.pack();
        maFenetre.setVisible(true);
    }
}
```

Redéfinition du gestionnaire avec bourrage.

Ajouts successifs des composants avec positionnement.

- NORTH
- WEST
- CENTER
- EAST
- SOUTH

À l'exécution :



¹<https://docs.oracle.com/javase/8/docs/api/java/awt/BorderLayout.html>

Le gestionnaire GridLayout

Ce gestionnaire¹ permet une disposition tabulaire des éléments :

```
import javax.swing.*;
import java.awt.*;

class PanneauInterieur extends JPanel {
    PanneauInterieur() {
        setLayout(new GridLayout(2, 2));
        JPanel langDescription = new JPanel();
        langDescription.add(new JLabel("HTML"));
        langDescription.add(new JLabel("UML"));
        langDescription.add(new JLabel("XML"));
        langDescription.setBorder(BorderFactory.createTitledBorder("Description"));
        add(langDescription);
        // Idem avec JPanel langFonctionnel, langObjet, langMulti.
        ...
    }
}

class ExempleGridLayout {
    public static void main(String args[]) {
        JFrame maFenetre = new JFrame("Avec GridLayout");
        maFenetre.setContentPane(new PanneauInterieur());
        maFenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        maFenetre.pack();
        maFenetre.setVisible(true);
    }
}
```

Redéfinition du gestionnaire (tableau 2x2).

Ajout d'une bordure avec titre pour chacun des sous-panneaux.

À l'exécution :



¹ <https://docs.oracle.com/javase/8/docs/api/java/awt/GridLayout.html>

Le gestionnaire GridBagLayout – description

Ce gestionnaire¹ permet une disposition tabulaire avec des cellules fusionnées.

✓ Grande souplesse

✗ Très technique

L'agencement sera décrit par des objets de type `GridBagConstraints`² :

- ▶ structurés par plus d'une dizaine de champs d'instance.
- ▶ dotés de plus d'une vingtaine de champs de classe.

Ce constructeur illustre cette finesse de paramétrage :

```
GridBagConstraints(int gridx, int gridy,  
                  int gridwidth, int gridheight,  
                  double weightx, double weighty,  
                  int anchor, int fill, Insets insets,  
                  int ipadx, int ipady)
```

● Coordonnées matricielles.

● Taille/Poids en largeur/hauteur du composant.

● Bourrage éventuel.

¹<https://docs.oracle.com/javase/8/docs/api/java/awt/GridBagLayout.html>

²<https://docs.oracle.com/javase/8/docs/api/java/awt/GridBagConstraints.html>

Le gestionnaire GridBagLayout – exemple très inspiré de l'API

```
import javax.swing.*;
import java.awt.*;

class PanneauInterieur extends JPanel {
    // Adaptation du code la page de l'API java (*)
    protected void creerBouton(String nom,
                               GridBagLayout gridBag,
                               GridBagConstraints c) {
        JButton bouton = new JButton(nom);
        gridBag.setConstraints(bouton, c);
        add(bouton);
    }

    PanneauInterieur() {
        GridBagLayout gridBag = new GridBagLayout();
        setLayout(gridBag);
        GridBagConstraints c = new GridBagConstraints();

        c.fill = GridBagConstraints.BOTH;
        c.weightx = 1.0;
        creerBouton("Bouton 1", gridBag, c);
        creerBouton("Bouton 2", gridBag, c);
        creerBouton("Bouton 3", gridBag, c);

        c.gridwidth = GridBagConstraints.REMAINDER;
        creerBouton("Bouton 4", gridBag, c);

        c.weightx = 0.0;
        creerBouton("Bouton 5", gridBag, c);

        c.gridwidth = GridBagConstraints.RELATIVE;
        creerBouton("Bouton 6", gridBag, c);
        ...
    }
}
```

* <https://docs.oracle.com/javase/8/docs/api/java/awt/GridBagLayout.html>

```
c.gridwidth = GridBagConstraints.REMAINDER;
creerBouton("Bouton 7", gridBag, c);

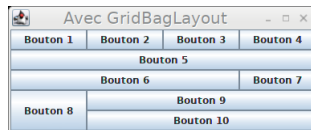
c.gridwidth = 1;
c.gridheight = 2;
c.weighty = 1.0;
creerBouton("Bouton 8", gridBag, c);

c.weighty = 0.0;
c.gridwidth = GridBagConstraints.REMAINDER;
c.gridheight = 1;
creerBouton("Bouton 9", gridBag, c);
creerBouton("Bouton 10", gridBag, c);
setSize(300, 100);
}

class ExempleGridBagLayout {
    public static void main(String args[]) {
        JFrame maFenetre = new JFrame("Avec GridBagLayout");
        maFenetre.setContentPane(new PanneauInterieur());
        maFenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        maFenetre.pack();
        maFenetre.setVisible(true);
    }
}
```



À l'exécution :



Le gestionnaire de répartition CardLayout – 1/2

Ce gestionnaire¹ permet :

- ▶ d'associer plusieurs composants graphiques à un même espace de l'IU ;
- ▶ de choisir le composant qui doit être vu. Ces composants seront gérés comme une pile.

On peut piloter l'affichage en utilisant des boutons dont on traitera les signaux :

```
import java.awt.event.*;

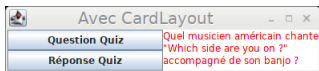
class MaClassAvecCardLayout
    implements ActionListener {
    ...
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == boutonQuestion) {
            gestCartes.show(ensCartes, etiquette);
        }
        ...
    }
}
```

Implémentation de l'interface ActionListener.

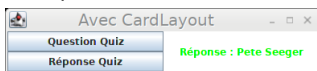
Usage de la méthode show().

À l'exécution, nous aurons en appuyant sur le bouton :

▶ Question Quiz



▶ Réponse Quiz



Pete Seeger : <https://www.youtube.com/watch?v=5iA1M02kv0g>

¹ <https://docs.oracle.com/javase/8/docs/api/java/awt/CardLayout.html>

Le gestionnaire de répartition CardLayout – 2/2

Code source du précédent exemple :

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

class CartesSuperposees
    extends JPanel
    implements ActionListener {
    JButton boutonQuestion = new JButton("Question Quiz");
    JButton boutonReponse = new JButton("Réponse Quiz");
    CardLayout gestCartes = new CardLayout();
    JPanel ensCartes = new JPanel();

    public CartesSuperposees() {
        this.setLayout(new GridLayout(1, 2));
        ensCartes.setLayout(gestCartes);
        // Première carte.
        String musique = "Quel musicien américain chante\n" +
            "\n\"Which side are you on ?\n\" +
            "accompagné de son banjo ?";
        JTextArea question = new JTextArea(musique);
        question.setForeground(Color.red);
        ensCartes.add(question, "une question");
        // Seconde Carte.
        JLabel reponse = new JLabel("Réponse : Pete Seeger",
            SwingConstants.CENTER);

        reponse.setOpaque(true);
        reponse.setBackground(Color.white);
        reponse.setForeground(Color.green);
        ensCartes.add(reponse, "la reponse");
        // Interaction avec les boutons.
        boutonQuestion.addActionListener(this);
        boutonReponse.addActionListener(this);
        ...
    }
}
```

```
JPanel panneau = new JPanel();
panneau.setLayout(new GridLayout(2, 1));
panneau.add(boutonQuestion);
panneau.add(boutonReponse);
// Ajout du panneau de JButton, puis des cartes.
add(panneau);
add(ensCartes);

}

public void actionPerformed(ActionEvent e) {
    Object obj = e.getSource();
    if (obj == boutonQuestion) {
        gestCartes.show(ensCartes, "une question");
    }
    else if (obj == boutonReponse) {
        gestCartes.show(ensCartes, "la reponse");
    }
}

public static void main(String args[]) {
    JFrame f = new JFrame("Avec CardLayout");
    f.setContentPane(new CartesSuperposees());
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.pack();
    f.setResizable(true);
    f.setVisible(true);
}
}
```

GridLayout permet de bien gérer l'étirement.

La répartition purement manuelle

Elle sera fortement déconseillée et réservée à des cas bien particuliers :

```
import javax.swing.*;
import java.awt.*;

class PanneauInterieur extends JPanel {
    PanneauInterieur() {
        JToggleButton pause, lecture;
        pause = new JToggleButton("Pause");
        lecture = new JToggleButton("Lecture");
        JLabel message;
        message = new JLabel("Placement manuel");
        setPreferredSize(new Dimension(500, 500));
        setLayout(null);
        pause.setSize(140, 90);
        pause.setLocation(100, 400);
        lecture.setSize(140, 90);
        lecture.setLocation(400, 100);
        message.setSize(300, 90);
        message.setLocation(300, 250);
        add(pause);
        add(lecture);
        add(message);
    }
}

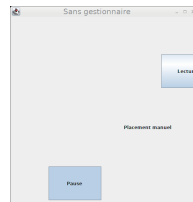
class ExempleSansLayout {
    public static void main(String args[]) {
        JFrame maFenetre = new JFrame("Sans gestionnaire");
        maFenetre.setContentPane(new PanneauInterieur());
        maFenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        maFenetre.pack();
        maFenetre.setVisible(true);
    }
}
```

Retirer le gestionnaire du composant avec `setLayout(null)`

Utiliser les méthodes :

- `setLocation()`
- `setLocationRelativeTo()`
- `setSize()`

À l'exécution, nous aurons :



Concepts de base

Une IU permet à l'utilisateur d'interagir avec le programme en agissant sur les composants de l'interface.

► Exemples d'interaction :

- action sur des composants :
 - changement de *focus*
 - insertion dans un champs
 - clic pression sur un bouton
 - clic sur les éléments d'une liste
 - ...
- clic de la souris ;
- frappe d'une touche au clavier.

► Ces interactions induisent des objets « événements » :

- FocusEvent
- InputEvent
- ActionEvent
- ItemEvent
- MouseEvent
- KeyEvent
- ...

tous dans le paquetage
`java.awt.event`.

Tout comme pour les exceptions, il est possible de scruter ces événements et de lancer un traitement particulier.

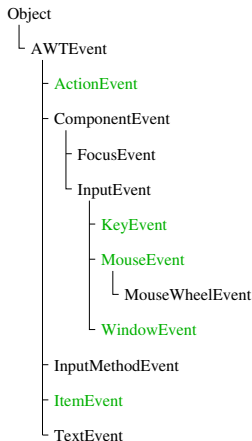
Notion d'écouteur

Un écouteur est un objet chargé de traiter les événements d'un certains types, et :

- implémente une interface `XXXListener` ($XXX \in \{Action, Mouse, Key, \dots\}$);
- redéfinit de façon appropriée ces méthodes.

Événements et objets écouteurs – programmation (1/2)

Lors d'une interaction (clavier, souris...), la JVM émet un événement d'un certain type¹ :



Pour qu'un objet écoute un certain type d'événements :

- 1 sa classe ABC doit implémenter une interface adaptée ;

```
class ABC implements XXXListener {...}
```

- 2 l'objet émetteur objEmtr doit être associé à cet objet écouteur ObjEctr.

```
objEmtr.addXXXListener(objEctr);
```

avec $XXX \in \{Action, Mouse, Key, \dots\}$

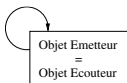
Les objets écoutant ces événements invoqueront toutes les méthodes nécessaires.

L'ordre d'évaluation n'est pas garanti !

¹<https://docs.oracle.com/javase/8/docs/api/java/awt/event/package-tree.html>

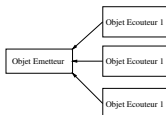
Événements et objets écouteurs – programmation (2/2)

Un objet composé d'objets émetteurs peut aussi être son propre écouteur.



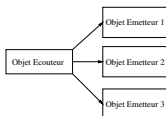
```
// Dans le constructeur (le plus souvent) :  
objEmtr.addXXXListener(this);  
// Ou bien, selon les cas,  
addXXXListener(this);
```

Un objet émetteur peut très bien être écouté par plusieurs objets.



```
// Dans le constructeur d'objEmtr (le plus souvent) :  
objEmtr.addXXXListener(objEctr1);  
...  
// Ou bien, selon les cas,  
addXXXListener(objEctr1);  
...
```

Un objet écouteur peut très bien écouter plusieurs émetteurs.



```
// Dans le constructeur d'objEmtr ou objEctr :  
objEmtr1.addXXXListener(objEctr);  
...  
// Ou bien, selon les cas, dans le constructeur d'objEmtr :  
addXXXListener(objEctr);  
...  
// NB : ces événements pouvant être de types différents.
```

Exemples : tracé de segments contrôlé par des boutons

Il existe de nombreuses façons de réaliser une interface utilisateur :

avec leurs pous

avec leurs contres

Nous exposons trois réalisations pour la gestion des événements :

- 1 IU et traitement des événements mélangés.

⇒ simple/⇒ vite illisible

- 2 Association entre chaque composant et ses événements.

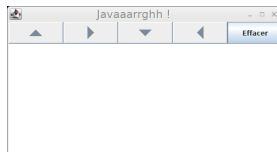
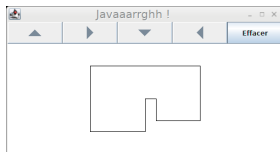
⇒ lisibilité/⇒ mélange

- 3 Séparation entre composants et la gestion des événements.

⇒ lisibilité/⇒ travail

Exemple : une zone de dessin et cinq boutons tels qu'une pression sur :

- ▶ l'une des flèches tracera un segment noir ;
- ▶ le bouton « Effacer » gommara tout.



Exemple : première implémentation – (1/3)

Première implémentation : les différents import et la méthode main().

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.plaf.basic.BasicArrowButton;
import static javax.swing.JFrame.*;
import static javax.swing.SwingConstants.*;
import static javax.swing.plaf.basic.BasicArrowButton.*;

class MobileBoutons_1 {
    public static void main(String args[]) {
        JFrame f = new JFrame("Javaaarrghh !");
        f.setContentPane(new DessinerTrajectoire());
        f.pack();
        f.setVisible(true);
        f.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

Permettra d'utiliser des boutons avec des flèches prédéfinies (héritant de la classe JButton).

Permettra d'alléger l'écriture du code.

Exemple : première implémentation – (2/3)

Seconde partie du code : la construction de l'interface.

```
class DessinerTrajectoire
    extends JPanel
    implements ActionListener {
    int tabDirection[] = {NORTH, EAST, SOUTH, WEST};
    BasicArrowButton tabBtnDir[];
    JButton effacer = new JButton("Effacer");
    JPanel zoneDessin = new JPanel();

    int positionX = 250;
    int positionY = 100;
    int increment = 20;

    DessinerTrajectoire() {
        setLayout(new BorderLayout(this, BorderLayout.Y_AXIS));
        JPanel panneauBouton = new JPanel();
        panneauBouton.setPreferredSize(new Dimension(270, 40));
        panneauBouton.setLayout(new GridLayout(1, 5));
        tabBtnDir = new BasicArrowButton[tabDirection.length];
        for (int i = 0; i < tabDirection.length; i++) {
            tabBtnDir[i] = new BasicArrowButton(tabDirection[i]);
            panneauBouton.add(tabBtnDir[i]);
        }
        panneauBouton.add(effacer);
        add(panneauBouton);
        zoneDessin.setPreferredSize(new Dimension(500, 200));
        zoneDessin.setBackground(Color.WHITE);
        add(zoneDessin);

        for (int i = 0; i < tabDirection.length; i++) {
            tabBtnDir[i].addActionListener(this);
        }
        effacer.addActionListener(this);
    }
}
```

Remarques :

Ajout d'une interface *ActionListener*.

Définition de champs pour les composants graphiques qui seront réutilisés dans le code.

Construction et ajouts des composants.

Ajouts de l'objet écouteur.

Exemple : première implémentation – (3/3)

Troisième partie du code : la gestion des événements.

```
public void actionPerformed(ActionEvent e) {  
    Object obj = e.getSource();  
    if (obj == effacer) {  
        zoneDessin.repaint();  
        positionX = 250;  
        positionY = 100;  
    }  
    else {  
        Graphics g = zoneDessin.getGraphics();  
        int direction = 0;  
        for (int i = 0; i < tabDirection.length; i++) {  
            if (obj == tabBtnDir[i]) {  
                direction = tabBtnDir[i].getDirection();  
                break;  
            }  
        }  
        int ancienX = positionX;  
        int ancienY = positionY;  
        switch (direction) {  
            case NORTH:  
                positionY -= increment;  
                break;  
            ...  
            case EAST:  
                positionX += increment;  
                break;  
        }  
        g.drawLine(ancienX, ancienY, positionX, positionY);  
        g.dispose();  
    }  
}
```

Remarques :

● Obtention de l'objet émetteur de l'objet de type `ActionEvent` « intercepté ».

● Appel à la méthode `repaint()`. Cette méthode de la classe `Component` redessinera correctement le panneau (avec le réglage de sa couleur de fond).

● Obtention du contexte graphique de l'objet `zoneDessin`.

⇒ simple / ⇒ vite illisible

Exemple : deuxième implémentation – association (1/2)

Construction de la fenêtre :

```
import java.awt.event.*;
...

class MobileBoutons_2 {
    public static void main(String args[]) {
        JFrame f = new JFrame("Javaaarrghh !");
        f.setContentPane(new InterieurEtParametres());
        ...
    }
}

class InterieurEtParametres extends JPanel {
    JPanel zoneDessin = new JPanel();
    int positionX = 250;
    int positionY = 100;
    int increment = 20;

    InterieurEtParametres() {
        setLayout(new BorderLayout(this, BorderLayout.Y_AXIS));
        JPanel panneauBouton = new JPanel();
        panneauBouton.setPreferredSize(new Dimension(270, 40));
        panneauBouton.setLayout(new GridLayout(1, 5));
        BoutonsDirections bd = new BoutonsDirections(zoneDessin, this);
        for (BasicArrowButton bouton : bd.tabBtnDir) {
            panneauBouton.add(bouton);
        }
        panneauBouton.add(new BoutonEffacer("Effacer", zoneDessin, this));
        add(panneauBouton);
        zoneDessin.setPreferredSize(new Dimension(500, 200));
        zoneDessin.setBackground(Color.WHITE);
        add(zoneDessin);
    }
}
```

Remarques :

Cette nouvelle classe se charge de la construction graphique et hébergera les paramètres nécessaires au fonctionnement.

Exemple : deuxième implémentation – association (2/2)

```
class BoutonsDirections
    extends JPanel
    implements ActionListener {

    InterieurEtParametres param;
    int tabDirection[] = {NORTH, EAST, SOUTH, WEST};
    BasicArrowButton tabBtnDir[];
    JPanel zoneDessin;

    BoutonsDirections(JPanel zd, InterieurEtParametres ip) {
        param = ip;
        tabBtnDir = new BasicArrowButton[tabDirection.length];
        for (int i = 0; i < tabDirection.length; i++) {
            tabBtnDir[i] = new BasicArrowButton(tabDirection[i]);
            ip.add(tabBtnDir[i]);
            tabBtnDir[i].addActionListener(this);
        }
        zoneDessin = zd;
    }

    public void actionPerformed(ActionEvent e) {
        ...
        int ancienX = param.positionX;
        ...
        switch (direction) {
            case NORTH:
                param.positionY -= param.increment;
                break;
            ...
        }
        ...
    }
}
```

Remarques : Définition de deux classes distinctes pour chacun des composants. Les composants gèrent eux-mêmes les événements.

⇒ lisibilité/⇒ mélange

```
class BoutonEffacer
    extends JButton
    implements ActionListener {

    InterieurEtParametres param;
    JPanel zoneDessin;

    BoutonEffacer(String n, JPanel zd,
                    InterieurEtParametres ip) {
        super(n);
        zoneDessin = zd;
        param = ip;
        addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        Object obj = e.getSource();
        if (obj == this) {
            zoneDessin.repaint();
            param.positionX = 250;
            param.positionY = 100;
        }
    }
}
```



Exemple : troisième implémentation – séparation (1/2)


```
import java.awt.event.*;
...

class MobileBoutons_3 {
    public static void main(String args[]) {
        ...
        f.setContentPane(new ConstructionIU());
        ...
    }
}


class ConstructionIU extends JPanel {
    DessinerTrajectoire() {
        setLayout(new BorderLayout(this,
                                BoxLayout.Y_AXIS));
        JPanel panneauBouton = new JPanel();
        panneauBouton.setPreferredSize(new Dimension(270, 40));
        panneauBouton.setLayout(new GridLayout(1, 5));
        int tabDirection[] = {NORTH, EAST, SOUTH, WEST};
        BasicArrowButton tabBtnDir[];
        tabBtnDir = new BasicArrowButton[tabDirection.length];
        for (int i = 0; i < tabDirection.length; i++) {
            tabBtnDir[i] = new BasicArrowButton(tabDirection[i]);
            panneauBouton.add(tabBtnDir[i]);
        }
        JButton effacer = new JButton("Effacer");
        panneauBouton.add(effacer);
        add(panneauBouton);
        JPanel zoneDessin = new JPanel();
        zoneDessin.setPreferredSize(new Dimension(500, 200));
        zoneDessin.setBackground(Color.WHITE);
        add(zoneDessin);
        ...
    }
}
```

Remarques :

le constructeur de cette classe effectue


 la construction de l'interface graphique;

puis


 la construction d'un objet écouteur connecté aux objets émetteurs.

```
Ecouteur objEcouteur = new Ecouteur(zoneDessin,
                                     tabDirection,
                                     tabBtnDir,
                                     effacer);

for (int i = 0; i < tabDirection.length; i++) {
    tabBtnDir[i].addActionListener(objEcouteur);
}
effacer.addActionListener(objEcouteur);
}
...
```


Exemple : troisième implémentation – séparation (2/2)

```
class Ecouteur implements ActionListener {
    BasicArrowButton tabBtnDir[];
    JButton effacer;
    JPanel zoneDessin;
    int tabDirection[];
    int positionX = 250;
    int positionY = 100;
    int increment = 20;

    Ecouteur(JPanel zd,
             int td[],
             BasicArrowButton tbd[],
             JButton e) {
        zoneDessin = zd;
        tabDirection = td;
        tabBtnDir = tbd;
        effacer = e;
    }

    // Méthode identique à celle de la première réalisation.
    public void actionPerformed(ActionEvent e) {
        Object obj = e.getSource();

        if (obj == effacer) {
            ...
        }
        else {
            Graphics g = zoneDessin.getGraphics();
            ...
        }
    }
}
```

Remarques :

Réalisation d'une classe Ecouteur :

- ▶ implémentant l'interface *ActionListener* ;
- ▶ et disposant comme champs

de références sur les composants graphiques émetteurs,

et des paramètres nécessaires pour le dessin.

⇒ lisibilité/⇒ travail

Conseils pour réaliser une interface utilisateur

Définir un modèle de données sous-jacent

+

Identifier les interactions

⇒ facile en apparence mais... très dure !

Lister les composants

+

Définir une disposition

⇒ partie quasi « cosmétique ».

Gérer correctement les interactions

⇒ dépend des choix de la 1^{re} partie...

L'interface *ActionListener* – (1/2)

C'est l'interface¹ la plus simple, la plus courante et donc la moins spécifique.

Le programmeur devra redéfinir la méthode :

```
void actionPerformed(ActionEvent e)
```

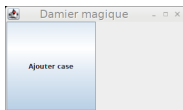
⇒ action quelconque

tout en s'appuyant sur l'objet `e` de type `actionEvent`² et le plus généralement les méthodes :

```
String getActionCommand();
long getWhen();
Object getSource(); // méthode héritée.
```

Exemple : composons une fenêtre avec un bouton à gauche et une zone à droite sur laquelle seront ajoutées progressivement les cases d'un damier.

Au début :



Pendant :



À la toute fin :



¹<https://docs.oracle.com/javase/8/docs/api/java/awt/event/ActionListener.html>

²<https://docs.oracle.com/javase/8/docs/api/java/awt/event/ActionEvent.html>

L'interface *ActionListener* – autre exemple (2/2)

```

1 import java.awt.*;
2 import javax.swing.*;
3 import java.awt.event.*;
4 import static javax.swing.JFrame.*;
5 import static java.awt.Color.*;
6 import static javax.swing.SwingConstants.*;
7
8 class DamierDynamique
9     extends JFrame
10     implements ActionListener {
11     int cptCase;
12     JPanel zoneDamier = new JPanel();
13     JButton boutonAjouterCase = new JButton("Ajouter case");
14
15     DamierDynamique() {
16         getContentPane().setLayout(new GridLayout(1, 2));
17         add(boutonAjouterCase);
18         add(zoneDamier);
19         zoneDamier.setLayout(new GridLayout(8, 8, 3, 3));
20         zoneDamier.setPreferredSize(new Dimension(8 * 100,
21                                                     8 * 100));
22         boutonAjouterCase.addActionListener(this);
23     }
24
25     ...

```

Remarques :

- ▶ Usage de différents import static
- ▶ Utilisation de la méthode validate()
- ▶ Algorithme de gestion de l'affichage de GridLayout peu intuitif...

```

25 public void actionPerformed(ActionEvent e) {
26     if ( (boutonAjouterCase == e.getSource()) &&
27         (cptCase < 64) ) {
28         JLabel c = new JLabel(" " + cptCase, CENTER);
29         c.setOpaque(true);
30         c.setSize(100, 100);
31         int pariteLigne = (cptCase / 8) % 2;
32         Color cFond, cEcriture;
33         if ( (cptCase + pariteLigne) % 2 == 0 ) {
34             cFond = WHITE;
35             cEcriture = BLACK;
36         }
37         else {
38             cFond = BLACK;
39             cEcriture = WHITE;
40         }
41         c.setBackground(cFond);
42         c.setForeground(cEcriture);
43         cptCase++;
44         zoneDamier.add(c);
45         zoneDamier.validate();
46     }
47 }
48
49 public static void main(String args[]) {
50     DamierDynamique damier = new DamierDynamique();
51     damier.setTitle("Damier magique");
52     damier.pack();
53     damier.setVisible(true);
54     damier.setDefaultCloseOperation(EXIT_ON_CLOSE);
55 }
56
57 }

```



La fenêtre : les interfaces *WindowListener* et *WindowFocusListener*

Ces interfaces¹ permettront de gérer finement le comportement de la fenêtre.

Le programmeur devra redéfinir ces sept méthodes :

<code>// Interface WindowListener</code>	
<code>void windowActivated(WindowEvent e)</code>	⇒ à l'activation de la fenêtre
<code>void windowClosed(WindowEvent e)</code>	⇒ après fermeture
<code>void windowClosing(WindowEvent e)</code>	⇒ lors de la fermeture
<code>void windowDeactivated(WindowEvent e)</code>	⇒ lorsque la fenêtre n'est plus active
<code>void windowDeiconified(WindowEvent e)</code>	⇒ lors de l'icônification
<code>void windowIconified(WindowEvent e)</code>	⇒ lors de la désicônification
<code>void windowOpened(WindowEvent e)</code>	⇒ lors de la première ouverture
<code>// Interface WindowFocusListener</code>	
<code>void windowGainedFocus(WindowEvent e)</code>	⇒ reprise du focus
<code>void windowLostFocus(WindowEvent e)</code>	⇒ perte du focus

tout en s'appuyant sur les informations fournies par l'objet `e` de type `WindowEvent`² :

- ▶ Nombreuses constantes statiques associées aux touches :

```
WINDOW_ACTIVATED
WINDOW_CLOSED
...
WINDOW_DEACTIVATED
WINDOW_ICONIFIED
```

- ▶ Méthodes pour analyser l'objet :

```
int getNewState();
int getOldState();
Window getWindow();
...
```

¹<https://docs.oracle.com/javase/8/docs/api/java/awt/event/WindowListener.html>

<https://docs.oracle.com/javase/8/docs/api/java/awt/event/WindowFocusListener.html>

²<https://docs.oracle.com/javase/8/docs/api/java/awt/event/WindowEvent.html>

Le clavier : l'interface *KeyListener*

Cette interface¹ permet de traiter les événements issus du clavier.

Le programmeur devra redéfinir ces trois méthodes :

```
void keyPressed(KeyEvent e);
void keyReleased(KeyEvent e);
void keyTyped(KeyEvent e);
```

⇒ touche enfoncée.

⇒ relâchement de touche.

⇒ frappe de touche.

tout en s'appuyant sur les informations apportées par l'objet *KeyEvent*² détecté :

► Nombreuses constantes statiques associées aux touches :

```
VK_0 // Touche 0 du pavé numérique.
VK_1
VK_2
...
VK_DEAD_MACRON // incroyable mais vrai !
```

► Méthodes pour analyser l'objet :

```
char getKeyChar();
int getKeyCode();
...
```

Focus d'entrée, *JPanel* et la méthode *setFocusable(true/false)*

De façon native, un objet *JFrame* se voit adresser les frappes du clavier. Pour qu'un objet *JPanel* puisse disposer aussi du focus du clavier, il faut l'activer avec *setFocusable(true)*.

¹<https://docs.oracle.com/javase/8/docs/api/java/awt/event/KeyListener.html>

²<https://docs.oracle.com/javase/8/docs/api/java/awt/event/KeyEvent.html>

La souris : l'interface *MouseListener* – (1/2)

Cette interface¹ permet de traiter les événements issus de la souris :

- ▶ divers clics ;
- ▶ mouvements.

Le programmeur devra redéfinir les méthodes :

```
void mouseClicked(MouseEvent e)
void mouseEntered(MouseEvent e)
void mouseExited(MouseEvent e)
void mousePressed(MouseEvent e)
void mouseReleased(MouseEvent e)
```

⇒ clic souris

⇒ entrée du pointeur dans un composant

⇒ sortie du pointeur d'un composant

⇒ pression bouton de souris

⇒ relâchement bouton de souris

tout en analysant les informations fournies par l'objet *e* de type *MouseEvent*² :

- ▶ Nombreuses constantes statiques associées aux touches.

```
BUTTON1 // cf. getButton()
BUTTON2
BUTTON3
...
MOUSE_PRESSED
...
```

- ▶ Méthodes pour analyser l'objet :

```
int getButton();
int getX();
int getY();
boolean isPopUpTrigger();
...
```

¹<https://docs.oracle.com/javase/8/docs/api/java/awt/event/MouseListener.html>

²<https://docs.oracle.com/javase/8/docs/api/java/awt/event/MouseEvent.html>

La souris : les interfaces *Mouse[Motion/Wheel]Listener* – (2/2)

Ces interfaces¹ permettent de traiter les événements issus de la souris :

- ▶ mouvements (glisser) ;
- ▶ usage de la roue de la souris ;

Le programmeur devra redéfinir les méthodes :

```
// Interface MouseMotionListener
void mouseDragged(MouseEvent e);
void mouseMoved(MouseEvent e);
// Interface MouseWheelListener
void mouseWheelMoved(MouseWheelEvent e);
```

⇒ glisser d'un composant
⇒ déplacement du curseur sans pression

⇒ action de la roue de la souris

tout en analysant sur les informations également fournies par l'objet *e* de type *MouseWheelEvent*² :

- ▶ Deux constantes statiques associées aux touches.

```
WHEEL_BLOCK_SCROLL
WHEEL_UNIT_SCROLL
```

- ▶ Méthodes pour analyser l'objet :

```
int getScrolAmount();
int getWheelRotation();
...
```

¹<https://docs.oracle.com/javase/8/docs/api/java/awt/event/MouseMotionListener.html>
<https://docs.oracle.com/javase/8/docs/api/java/awt/event/MouseWheelListener.html>

²<https://docs.oracle.com/javase/8/docs/api/java/awt/event/MouseWheelEvent.html>

Les listes : l'interface *ItemListener*

Cette interface¹ permet de traiter les événements issus de la sélection/dé-sélection d'entrée dans une liste :

Le programmeur devra redéfinir l'unique méthode :

```
void itemStateChanged(ItemEvent e)
```

⇒ sélection/dé-sélection d'une entrée

tout en analysant les informations fournies par l'objet `e` de type `ItemEvent`² :

- Constantes statiques associées :

```
DESELECTED
ITEM_FIRST
ITEM_LAST
ITEM_STATE_CHANGED
SELECTED
```

- Méthodes pour analyser l'objet :

```
Object getItem() // -> Item affecté.
int getStateChange() // -> [de/se]lected.
...
```

¹<https://docs.oracle.com/javase/8/docs/api/java/awt/event/ItemListener.html>

²<https://docs.oracle.com/javase/8/docs/api/java/awt/event/ItemEvent.html>